



Práctica 4:

Mezclando C y ensamblador



Objetivos principales

- Conocer la diferencia entre variables **locales y globales**.
- Analizar los problemas que surgen cuando se utilizan **varios ficheros fuentes** en un mismo proyecto, y entender cómo **se resuelven los símbolos**.
- Comprender la relación entre **nuestro programa C** y el **código máquina** generado por un compilador *gcc*.
- Ser capaz de utilizar **variables y funciones definidas en C desde un código ensamblador**, y viceversa.
- Tener un contacto inicial con la **representación de tipos de datos estructurados** en lenguajes de alto nivel.

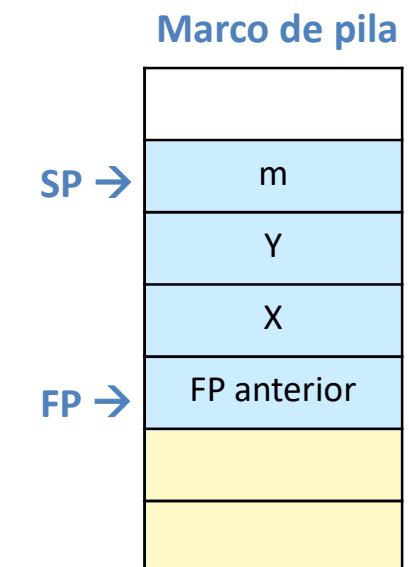


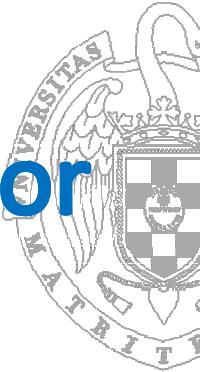
Traducir funciones de C a ensamblador

Uso de variables locales en ensamblador (ref. page 3)

```
int Mayor(int X, int Y){  
    int m;  
    if (X > Y)  
        m = X;  
    else  
        m = Y;  
    return m;  
}
```

```
1 Mayor:  
2     push {fp}  
3     mov fp, sp  
4     sub sp, sp, #12 @ reserva espacio para X, Y y m  
// X se almacena en fp-4, Y en fp-8 and m en fp-12  
5     str r0, [fp,#-4] @ inicializa X con el primer parámetro  
6     str r1, [fp,#-8] @ inicializa Y con el segundo parámetro  
7  
8     @ if( X > Y )  
9     ldr r0, [fp,#-4] @ r0 ← X  
10    ldr r1, [fp,#-8] @ r1 ← Y  
11    cmp r0, r1  
12    ble ELS  
13    @ then  
14    ldr r0, [fp,#-4] @ m = X;  
15    str r0, [fp,#-12]  
16    b RET  
17    @ else  
18 ELS:  
19    ldr r0, [fp,#-8] @ m = Y;  
20    str r0, [fp,#-12]  
21    @ return m  
22 RET:  
23    ldr r0, [fp,#-12] @ return value  
24    pop{fp}  
25    mov pc, lr
```

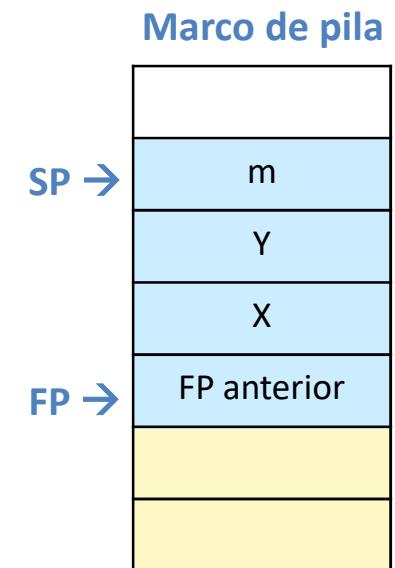




Traducir funciones de C a ensamblador

Uso de variables locales en ensamblador (ref. página 3)

```
// Una versión más legible de la misma función
. equ X, -4
. equ Y, -8
. equ m, -12
1 Mayor:
2   push {fp}
3   mov fp, sp
4   sub sp, sp, #12
5   str r0, [fp,#X] @ inicializa X con el primer parámetro
6   str r1, [fp,#Y] @ inicializa Y con el segundo parámetro
7
8   @ if( X > Y )
9   ldr r0, [fp,#X] @ r0 ← X
10  ldr r1, [fp,#Y] @ r1 ← Y
11  cmp r0, r1
12  ble ELS
13  @ then
14  ldr r0, [fp,#X] @ m = X;
15  str r0, [fp,#m]
16  b RET
17  @ else
18 ELS:
19  ldr r0, [fp,#Y] @ m = Y;
20  str r0, [fp,#m]
21  @ return m
22 RET:
23  ldr r0, [fp,#m] @ return value
24  pop{fp}
25  mov pc, lr
```





Optimizaciones del compilador

Compilando el mismo código con diferentes niveles de optimización
(ref. página 5)

Compilation without optimizations.
gcc -O0

```
#define N 10
int A[N];
int B[N];
int C,D,i;

int main(void) {
    for (i=0;i<N-1;i+=2) {
        A[i] = B[i] + C;
        A[i+1] = B[i] - D;
    }
    return 0;
}
```

Compilation with optimizations.
gcc -O2

```
main:
    push {fp}
    add fp, sp, #0
    ldr r3, =i
    mov r2, #0
    str r2, [r3]
    b COND
LOOP:
    ldr r3, =i
    ldr r2, [r3]
    ldr r3, =i
    ldr r1, [r3]
    ldr r3, =B
    ldr r1, [r3, r1, lsl #2]
    ldr r3, =C
    ldr r3, [r3]
    add r1, r1, r3
    ldr r3, =A
    str r1, [r3, r2, lsl #2]
    ldr r3, =i
    ldr r3, [r3]
    add r2, r3, #1
    ldr r3, =i
    ldr r1, [r3]
    ldr r3, =B
    ldr r1, [r3, r1, lsl #2]
    ldr r3, =D
    ldr r3, [r3]
    sub r1, r1, r3
    ldr r3, =A
    str r1, [r3, r2, lsl #2]
    ldr r3, =i
    ldr r3, [r3]
```

main:

```
ldr r3, =C
push {r4, r5, r6}
ldr ip, =A          @ ip <- A
ldr r6, [r3]         @ r6 <- C
ldr r3, =D
ldr r4, =B
ldr r5, [r3]         @ r5 <- D
mov r2, ip          @ r2 <- &A[0]
mov r3, #0          @ r3 es i*4
LOOP:
    ldr r1, [r4, r3] @ r1 <- B[i]
    add r0, r6, r1
    str r0, [ip, r3] @ A[i] <- r0
    add r3, r3, #8   @ 4*i <- 4*(i + 2)
    sub r1, r1, r5
    cmp r3, #40
    str r1, [r2, #4] @ *(r2+1) = r1
    add r2, r2, #8   @ salta 2 enteros
    bne LOOP
    ldr r3, =i
    mov r2, #10
    str r2, [r3]
    mov r0, #0
    pop {r4, r5, r6}
    bx lr
```

A[i]= B[i] + C
A[i+1]= B[i] - D



Acceso a datos de tamaño byte

Nuevas instrucciones máquina para almacenar / cargar bytes (ref. página 6)

- **LDRB**: load de un entero sin signo de tamaño byte. Al escribirse el dato sobre el registro destino se extiende con ceros hasta los 32 bits.
- **STRB**: se escribe en memoria un entero de tamaño byte obtenido de los 8 bits menos significativos del registro fuente.
- No hay restricciones en la dirección: las restricciones de alineamiento no son aplicables.

Algunos ejemplos:

LDRB R5, [R9]

@ It loads into the 8 least significant bits of R5, 8 bits
@ from Mem(R9), and resets the remaining bits of R5.

LDRB R3, [R8, #3]

@ It loads into the 8 least significant bits of R3, 8 bits from
@ Mem(R8+3), and resets the remaining bits of R3.

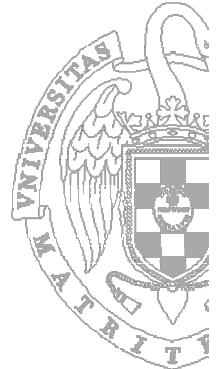
STRB R4, [R10, #0x200]

@ It stores into Mem(R10+0x200) the 8 least significant bits of R4.

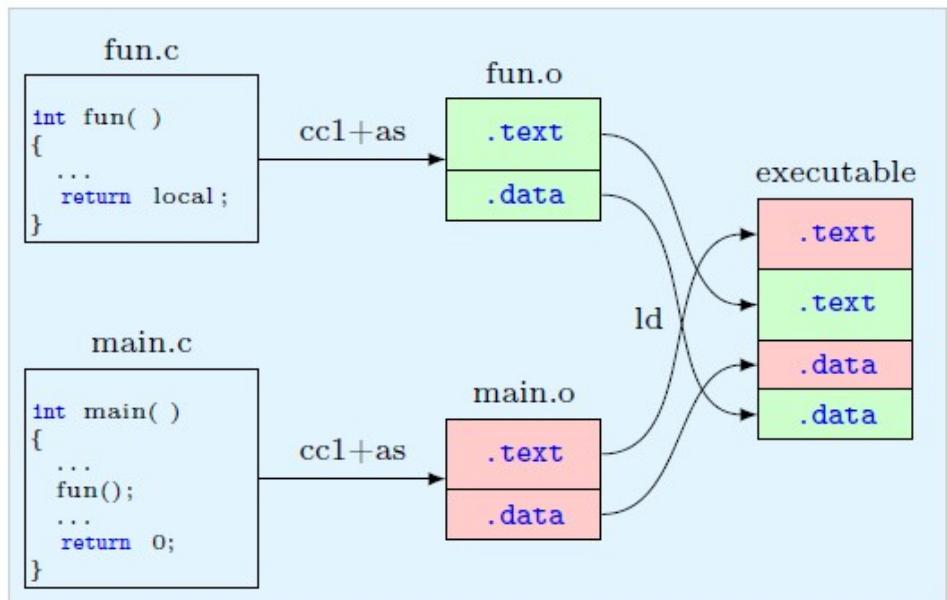
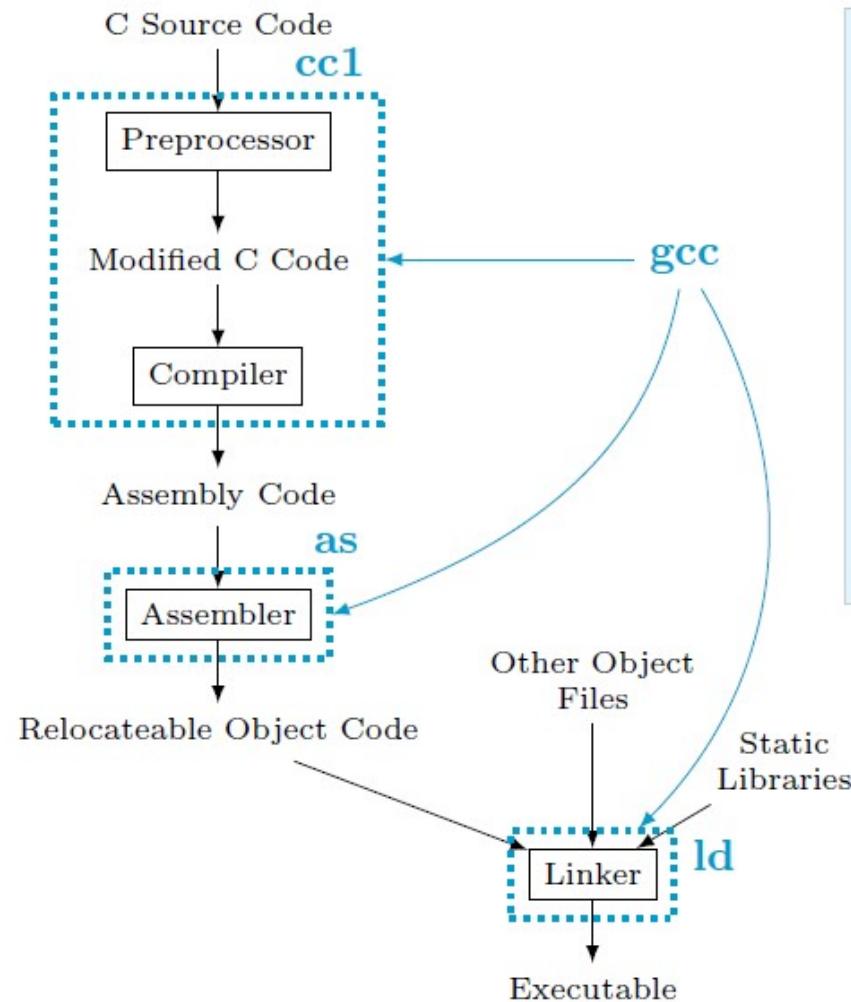
STRB R10, [R7, R4]

@ It stores into Mem(R7+R4) the 8 least significant bits of R10.

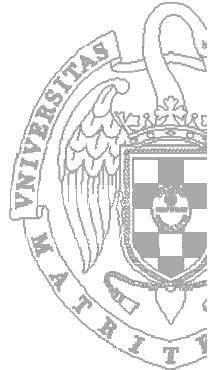
Combinar varias fuentes



Varias funciones escritas en C y en ensamblador pueden combinarse para crear un único ejecutable (.elf) (ref. página 6)



- ¿Qué es un “símbolo externo”?
 - ¿Como se resuelven las referencias a símbolos externos?
 - ¿Qué es la tabla de símbolos?
 - ¿Qué es el mapa de memoria?



Símbolos globales en C

Uso de funciones y variables externas en C (ref. páginas 8-9)

- En lenguaje C, **todas las funciones y variables globales se exportan como símbolos globales** por defecto.
- Para **utilizar una función que se define en otro archivo**, debe realizarse una declaración adelantada de la función.
 - Por ejemplo, para utilizar la función FOO, definida en otro archivo, que no recibe ni devuelve ningún parámetro, debemos usar la siguiente declaración adelantada antes de invocarla:
extern void FOO(void);
- En el caso de **variables globales**, también se debe hacer una declaración adelantada para usar una variable definida en otro archivo. Por ejemplo,
extern int aux;



Símbolos globales en ensamblador

Declaración y uso de símbolos globales en ensamblador (ref. página 9-10)

- A diferencia de C, en lenguaje ensamblador los símbolos son locales por defecto, es decir, invisibles desde otro archivo. Para convertirlos en símbolos globales debemos exportarlos usando la directiva `.global`.
- Además, cuando el programador desea utilizar un símbolo definido en otro archivo, debe utilizarse la directiva `.extern`.
- Ejemplo:

```
.extern FOO;      @ an extern symbol is made visible.  
.global start;   @ a local symbol is exported.
```

```
start:
```

```
    bl FOO;
```

```
...
```



Lanzar un programa C

En nuestro sistema (sin ningún sistema operativo) necesitamos un mecanismo para inicializar la pila y ejecutar la función “main” (ref. página 13)

Cuadro 2. Ejemplo de rutina de inicialización

```
.extern main
.extern _stack
.global start

start:
    ldr sp,=_stack
    mov fp,#0

    bl main

End:
    b End
.end
```

Fichero “init.s”



Depurar un programa C en Eclipse

Tenemos herramientas adicionales para tratar con programas escritos en C: las ventanas de Expresiones y Variables (ref. página 16)

The screenshot shows the Eclipse IDE interface during a C program debug session. The title bar indicates 'Debug - pract4b_15/main.c - Eclipse'. The menu bar includes File, Edit, Source, Refactor, Navigate, Project, Run, Window, Help. The toolbar has various icons for file operations, search, and run. The left pane shows the project structure with 'pract4b_15' selected, containing files like 'main.c', 'rutas.asm.asm', 't1.h', 'pract4b_15.map', and 'init.asm'. The main workspace displays the source code of 'main.c':

```
rutinas.asm.asm main.c t1.h pract4b_15.map init.asm
}
}

int main(void) {
    // 1. Crear una matriz NxM a partir del array lena128
    initRGB(imagenRGB);

    // 2. Transformar la matriz RGB a una matriz de grises
    RGB2GrayMatrix(imagenRGB, imagenGris);

    // 3. Transformar la matriz de grises a una matriz en blanco y negro
    Gray2BinaryMatrix(imagenGris, imagenBinaria);

    // 4. Contar los blancos que aparecen por filas en imagenBinaria
    contarBlancos(imagenBinaria, blancosPorFila);

    return 0;
}
```

The right side of the interface contains several debug windows:

- Variables**: Shows a list of variables with their current values. For example, 'blancosPorFila[0]' is 42, 'blancosPorFila[1]' is 39, etc.
- Breakpoints**: Shows the current breakpoints in the code.
- Expressions**: Allows evaluation of expressions at the current context.
- Registers**: Shows the state of CPU registers.
- Disassembly**: Shows the assembly code corresponding to the current instruction.
- Memory**: A hex dump viewer showing memory contents at address 0x001C000.

A large blue arrow on the left points towards the 'Variables' and 'Expressions' toolbars with the text 'Explicaciones adicionales usando Eclipse'.



Desarrollo de la sesión de laboratorio

Dada una imagen en color compuesta de 128x128 pixels, obtener una versión en escala de grises y una imagen binaria pura, donde cada punto se representa con negro o blanco (ref. página 18)





Estructura de la imagen en memoria

Está declarada como un array de píxeles. Cada píxel es una “struct” compuesta de tres bytes sin signo (ref. página 18)

```
typedef struct _pixel_RGB_t {  
    unsigned char R;  
    unsigned char G;  
    unsigned char B;  
} pixelRGB;
```

```
#define N 128  
#define M 128  
// defining the color image  
pixelRGB imagenRGB[N] [M];  
// defining the gray image  
unsigned char imagenGris [N] [M];
```

Píxeles y Colores: Ejemplos

R	G	B
79	129	189

218	102	50
-----	-----	----

70	246	22
----	-----	----

225	137	127
-----	-----	-----

imagenRGB [0] [0]



Inicialización de la imagen RGB

La entrada de datos es “lena128 []”, que es un vector de 3×2^{14} elementos.

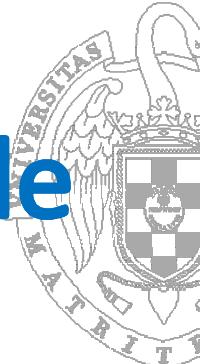
Cada elemento de lena128 es un byte (unsigned char en C).

Cada tripleta de lena128 representa un pixel de la imagen en color.

```
unsigned char lena128[] = {  
    225, 137, 127, 226, 135, 123, 227, 136, 123, 223, 134, 116, ... }
```

Las primeras 128 tripletas representan la primera fila de RGBImage, las siguientes 128 la segunda y así sucesivamente...

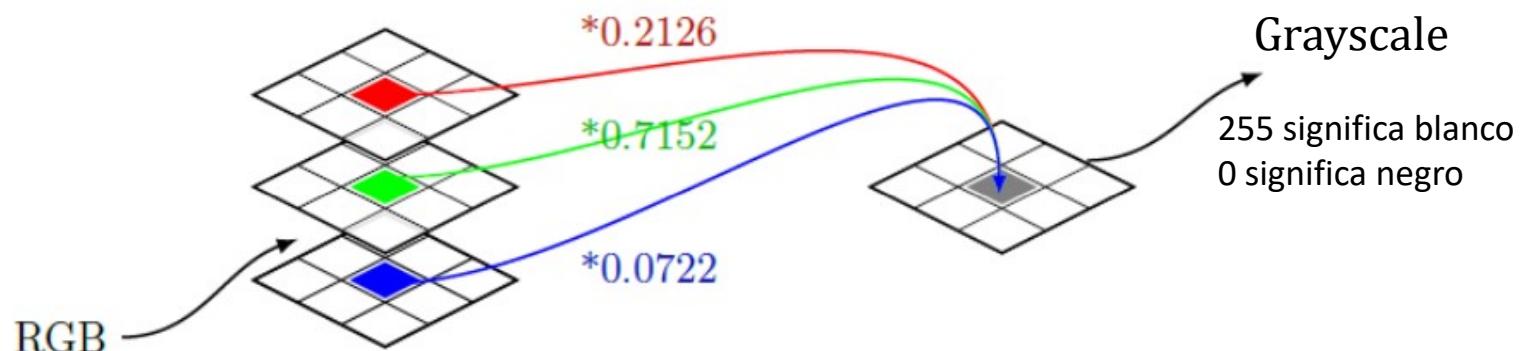
```
// A C function to initialize the RGBImage  
void initRGB(pixelRGB m[N][M]) {  
    int i,j;  
  
    for (i=0;i<N;i++)  
        for (j=0; j<M; j++) {  
            m[i][j].R = lena128[(i*M + j)*3];  
            m[i][j].G = lena128[(i*M + j)*3 + 1];  
            m[i][j].B = lena128[(i*M + j)*3 + 2];  
        }  
}
```



Transformando de RGB a grayscale

El valor de gris para un píxel en color dado se puede calcular de la siguiente manera (ref. página 19):

```
gray = 0.2126*pixel.R + 0.7152*pixel.G + 0.0722*pixel.B;
```



- Sin embargo, esto implica aritmética de punto flotante. Para evitarla podemos multiplicar las constantes por un número suficientemente grande, HN, y finalmente dividir el resultado por HN.
- Además, si elegimos HN como potencia de dos, la división se puede hacer sólo con desplazamientos. Por lo tanto, implementamos el siguiente cálculo:

```
gray = (3483*orig.R + 11718*orig.G + 1183*orig.B) / 16384;
```



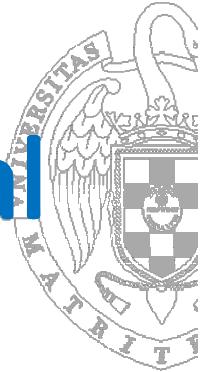
Transformando la imagen a binario

Definimos un número intermedio (umbral) entre 0 y 255 (ref. página 19)

Los tonos grises por encima del umbral se transforman en "blanco". Los que están por debajo (o iguales) al umbral se transforman en "negro"

Hemos definido el umbral como 127

```
for (i=0; i<N; i++)
    for (j=0; j<M; j++)
        if (imagenGris[i][j] > threshold)
            imagenBinaria[i][j]= 255;
        else
            imagenBinaria[i][j]= 0;
```



Estructura del programa principal

(ref. página 20)

```
int main(void) {
    // 1. Crear una matriz NxM a partir del array lena128
    initRGB(ImagenRGB);

    // 2. Transformar la matriz RGB a una matriz de grises
    RGB2GrayMatrix(ImagenRGB, imagenGris);

    // 3. Transformar la matriz de grises a una matriz B&N
    gray2BinaryMatrix(imagenGris, imagenBinaria);

    // 4. Contar los píxeles blancos en cada fila de imagenBinaria
    contarBlancos(imagenBinaria, blancosPorFila);

    return 0;
}
```



Asignación en mem de datos y código

Siempre podemos encontrar las direcciones que el enlazador ha asignado a cada función y estructura de datos analizando el contenido del archivo `<project_name>.map`, que está en la carpeta "Debug".

Prac4.map

Allocating common symbols

<u>Common symbol</u>	<u>size</u>	<u>file</u>
imagenBinaria	0x4000	./main.o
imagenGris	0x4000	./main.o
imagenRGB	0xc000	./main.o
blancosPorFila	0x80	./main.o



Asignación en mem de datos y código

(Continuación)

.data	0x0c000000	0xc000
	Start Addr.	size
* (.data)		
.data	0x0c000000	0x0 ./init.o
.data	0x0c000000	0xc000 ./lena128.o
	0x0c000000	lena128
.data	0x0c00c000	0x0 ./main.o
.data	0x0c00c000	0x0 ./routines_asm.o
.data	0x0c00c000	0x0 ./trafo.o
.....		
.....		
* (COMMON)		
COMMON	0x0c00c000	0x14080 ./main.o
	0x0c00c000	imagenBinaria
	0x0c010000	imagenGris
	0x0c014000	imagenRGB
	0x0c020000	blancosPorFila



Asignación en mem de datos y código

(Continuación)

	Start Addr.	size
.text	0x0c020080	0x51c
* (.text)		
.text	0x0c020080	0x14 ./init.o
	0x0c020080	start
.text	0x0c020094	0x0 ./lena128.o
.text	0x0c020094	0x1ec ./main.o
	0x0c020094	initRGB
	0x0c020228	main
.text	0x0c020280	0x38 ./routines_asm.o
	0x0c020280	rgb2gray
.text	0x0c0202b8	0x2e4 ./trafo.o
	0x0c0202b8	RGB2GrayMatrix
	0x0c020400	gray2BinaryMatrix
	0x0c0204e0	contarBlancos